Attorney Docket No.: 12407-004001

APPLICATION

FOR

UNITED STATES LETTERS PATENT

TITLE: BRANCH HANDLING FOR SINGLE INSTRUCTION MULTIPLE DATAPATH PROCESSOR ARCHITECTURES

APPLICANT: JOHN REDFORD

CERTIFICATE OF MAILING BY EXPRESS MAIL

Express Mail Label No. EL485518845

I hereby certify under 37 CFR §1.10 that this correspondence is being deposited with the United States Postal Service as Express Mail Post Office to Addressee with sufficient postage on the date indicated below and is addressed to the Commissioner for Patents, Washington, D.C. 20231.

November 28, 2000
Date of Deposit

Samantha Bell
Signature

Samantha Bell
Typed or Printed Name of Person Signing Certificate

Docket No.: 12407-004

Sub A1

BRANCH HANDLING FOR SINGLE INSTRUCTION MULTIPLE DATAPATH PROCESSOR ARCHITECTURES

CROSS-REFERENCE TO RELATED APPLICATIONS

This application is related to a copending application entitled HANDLING CONDITIONAL PROCESSING IN A SINGLE INSTRUCTION MULTIPLE DATAPATH PROCESSOR ARCHITECTURE, which was filed on the same day as this application and is incorporated herein by reference.

TECHNICAL FIELD

This invention relates to handling conditional processing in a single instruction multiple datapath (SIMD) processor architecture.

BACKGROUND

Parallel processing is an efficient way of processing an array of data items. A SIMD processor is a parallel processor array architecture wherein multiple datapaths are controlled by a single instruction. Each datapath handles one data item at a given time. In a simple example, in a SIMD processor having four datapaths, each data item in a four data item array would be processed in a respective one of the four datapaths.

During program execution in the SIMD processor, multiple datapaths may be enabled prior to encountering a conditional processing block, such as an if-then-else-processing block. Before executing the conditional processing block, the processor enable (PE) states of each of the datapaths, i.e., whether they are enabled or disabled, must be saved in case any of the datapath PE states is changed during execution of the conditional processing block. Further, upon exiting the conditional processing block, the PE states of the datapaths must be restored to the states that existed prior to entry of the conditional processing block.

SUMMARY

In a general aspect, the invention features a method of determining whether datapaths executing a computer program should execute conditional processing in the computer program. The method includes determining whether PE states of all of the datapaths are disabled, and branching around the conditional processing if the PE states of all of the

datapaths are disabled. Instructions are also provided for performing the determining and the branching.

In a preferred embodiment, branching is not performed if the program is determined to be deterministic.

The determination of whether the PE states of all of the datapaths are disabled includes evaluating a processor enable bit associated with each one of the datapaths. The processor enable bit is enabled if it is a value of one. The processor enable bit is disabled if it is a value of zero.

The determination of whether the computer program is deterministic includes evaluating a deterministic bit. The deterministic bit is a first value to indicate that the computer program is deterministic, and is a second value to indicate that the computer program is non-deterministic.

In another aspect, the invention features instructions that combine the branching with operations that maintain the PE states during the conditional processing. One such instruction causes the datapaths to establish a state of the datapaths' PE states for the conditional processing, determine whether the established PE states are all disabled, and branch around the conditional processing if the established PE states of all of the datapaths are disabled.

The conditional processing is, e.g., an if-processing block, and in this case the instructions also cause the datapaths to save a current state of the PE states prior to establishing them for the conditional processing. The conditional processing may also include an else-processing block.

Embodiments of various aspects of the invention may have one or more of the following advantages.

If all datapaths are disabled prior to entering an if-processing block or an else-processing block, there is no work to be accomplished in these blocks. Therefore, branching around the work allows the program to run faster. Combining the branching operation with operations that maintain the PE states during conditional processing provides faster, more efficient program execution, and simpler programming.

Testing a deterministic indicator provides a manner of overriding the branching in program code that must meet real time deadlines.

Other features and advantages of the invention will be apparent from the description and drawings, and from the claims.

DESCRIPTION OF DRAWINGS

FIG. 1 is a block diagram of a single instruction multiple datapath (SIMD) processor.

FIG. 2 is a block diagram of a program having branch processes for skipping conditional processing in some situations.

Like reference symbols in the various drawings indicate like elements.

DETAILED DESCRIPTION

Referring to FIG. 1, a single instruction multiple datapath (SIMD) processor 10 includes an instruction cache 12, control logic 14, a serial datapath 16, and a number of parallel datapaths labeled 18a, 18b, 18c, 18, ... 18n. The parallel datapaths 18 write to a memory 20. Each of the datapaths 18 has an associated processor enable (PE) bit 22 that represents the PE state of that datapath. Specifically, parallel datapath 18a is associated with a PE bit 22a, parallel datapath 18b is associated with a PE bit 22b, and so forth. When a PE bit is enabled, its associated parallel datapath is enabled and data items may be written by that parallel datapath. For example, if PE bit 22a is enabled, data items may be written by parallel datapath 18a; if PE bit 22b is enabled, data items may be written by parallel datapath 18b. If PE bit 22n is enabled, data items may be written by parallel datapath 18n. When a PE bit is disabled, its associated parallel datapath is disabled and data items may not be written by that parallel datapath.

In operation, the control logic 14 fetches an instruction from the instruction cache 12. The instruction is fed to the serial datapath 16 that provides the instruction to the datapaths 18. Each of the datapaths 18 is read together and written together unless the processor enable bit is disabled for a particular datapath.

When an instruction causes the SIMD processor 10 to execute a conditional processing block within the program code (e.g., a processing block that includes one or more if-then-else processing statements), the current PE state of each of the datapaths must be accounted for, so that if any of the PE states of the datapaths are altered during execution of the conditional processing block, the PE states can be restored upon the completion of the conditional processing block. Often, a conditional processing block contains multiple

conditional processing operations, some of which may be executed during (i.e., nested within) the processing of other conditional processing operations. In order to assure proper operation, the PE state of each datapath must be saved prior to entering each nested conditional operation, and the saved PE state must be restored upon completing the conditional operation.

Referring to Fig. 2, program code 40 that contains a conditional processing block 42 is shown. Conditional processing block 42 is an if-then-else processing block in this example, and thus includes an if-processing block 44 followed by an else-processing block 46. It will be understood that program code 40 may contain many other conditional processing blocks 42, and indeed, conditional processing block may include additional if-processing blocks 44 and/or else-processing blocks 46 nested within it. A single conditional processing block with one if-processing block 44 and one else-processing block 46 is shown merely for simplicity in describing an embodiment of the invention.

At the start of if-processing block 44, the current PE states of datapaths 18 are saved, and an if-processing statement is executed. The PE states of datapaths 18 are then set to either the enable state (PE=1) or the disable state (PE=0) according to the results of the if-processing statement. Only those datapaths having an enabled PE state will perform subsequent processing in if-processing block 44. Accordingly, if all datapaths 18 are set to the disabled PE state, no processing work will be performed within if-processing block 44.

Branch process 50 is inserted in program code 40 at the start of if-processing block 44. Branch process 50 tests the PE states of datapaths 18 upon the execution of the if-processing statement. If all datapaths are disabled (i.e., PE=0), the processing operation in if-processing block 44 may be skipped without affecting the computational results of program 40. Accordingly, branch process 50 branches 52 around if-processing block 44 to else-processing block 46.

The PE states of datapaths 18 are also tested at the start of else-processing block 46. If all datapaths 18 are disabled (i.e., PE=0), no processing work will be performed within else-processing block 46. Accordingly, branch process 60 branches 62 around else-processing block 46 to, in this example, the end 64 of conditional processing block 40.

In some cases, the execution of program 40 is deterministic. That is, for one reason or another (such as to meet real-time deadlines), it is desirable to execute program 40 in the

same amount of time regardless of whether any work in the program (such as if-processing block 44 and/or else-processing block 46) could be skipped. If so, a deterministic bit (DET, Fig. 1) is set by the programmer in a control register of SIMD processor 10. Branch processes 50, 60 test the state (0 or 1) of the DET bit and do not branch 52, 62 if the DET bit is set.

The copending application describes methods of saving and maintaining the PE states of datapaths 18 during conditional processing, such as if-then-else processing. Branch processes 50, 60 combine the branching determination with the PE state setting and maintaining operations, to provide instructions that respectively handle all of the work needed for an if-processing statement and an else-processing statement.

Branch process 50 combines the PE state saving operation with the branching operation, and is of the following form:

if (SAVE_PE (Px), PE = Pn =0) go to X

Branch process 50 saves the PE state of the datapaths in register Px, and then sets the PE state equal to the contents of register Pn. If those contents are 0 (i.e., if none of the datapaths' PE bits are set), branch process 50 branches 52 to destination X (e.g., the subsequent else-processing block 46).

As described in the copending application, a datapath's PE state is, under some conditions, inverted (i.e., from 0 to 1 or 1 to 0) prior to an else-processing block. The instruction for doing so is called a "FLIP" instruction in the copending application. Branch process 60 combines the FLIP instruction with the branching operation, and is of the following form:

if (FLIP_PE (Px)) go to Y

Branch process 60 will invert the appropriate PE bits (according to the rules described in the copending application) and will branch to destination Y (i.e., the end 64 of else-processing block 46) if none of the PE bits are set.

Other embodiments are within the scope of the following claims.

For example, branch processes 50, 60 may be used with other instructions that save and manipulate PE states during conditional processing.

Another branch process may be inserted at the start of conditional processing block 42 to determine whether the current PE states of all datapaths 18 (i.e. the PE states prior to

performing any conditional processing statement) are disabled. If all PE states are disabled, none of datapaths 18 will perform either if-processing block 44 or else-processing block 46. Accordingly, this branch process will branch around the entire conditional processing block 42. As with branch processes 50, 60, branching is not performed if the deterministic bet DET is set.

Embodiments of the invention have been described. Nevertheless, it will be understood that various modifications may be made without departing from the spirit and scope of the invention. Accordingly, other embodiments are within the scope of the following claims.